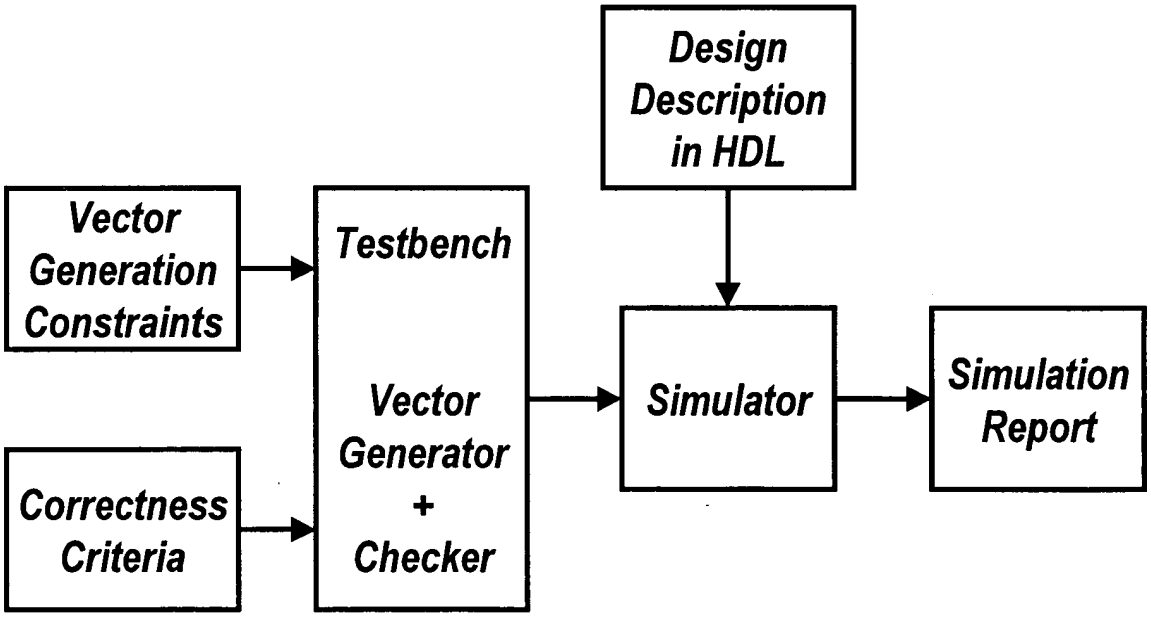
Design Description in HDL
Vector Generation Constraints
Testbench
Vector Generator + Checker
Correctness Criteria
Simulator
Simulation Report

Figure 1

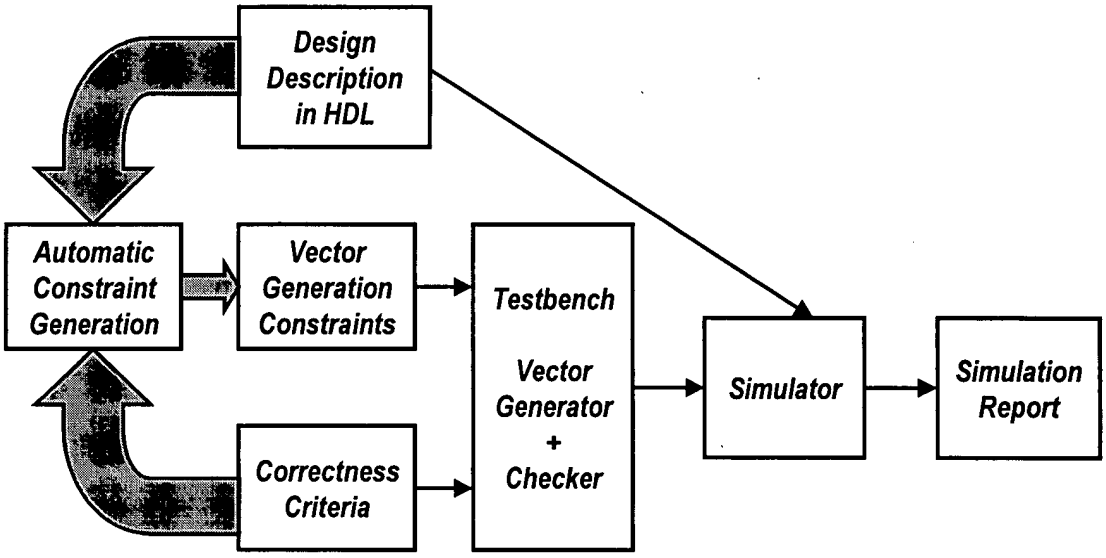
Design Description in HDL
Automatic Constraint Generation
Vector Generation Constraints
Testbench
Vector Generator + Checker
Correctness Criteria
Simulator
Simulation Report

Figure 2

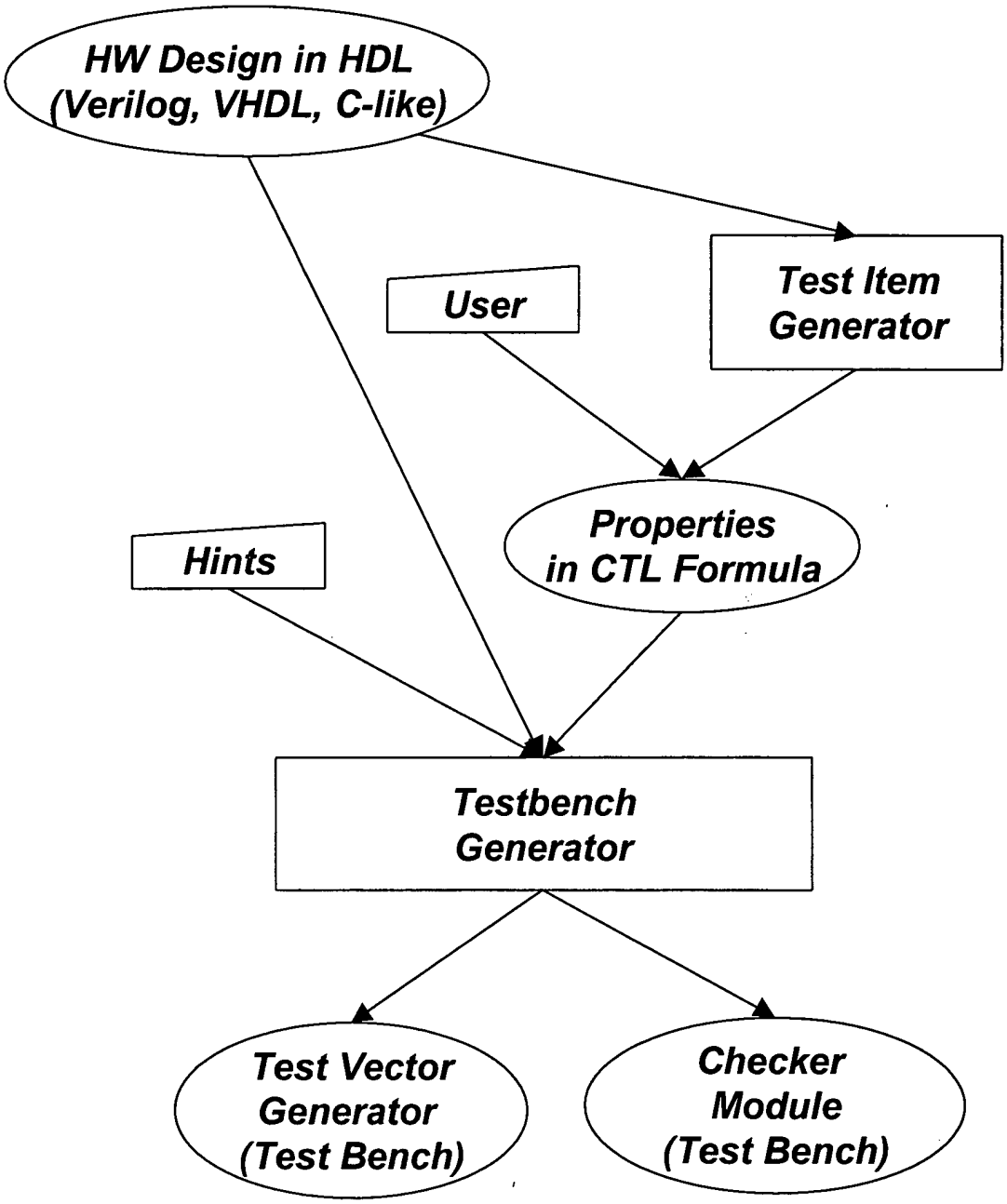
HW Design in HDL
(Verilog, VHDL, C-like)
User
Test Item
Generator
Hints
Properties
in CTL Formula
Testbench
Generator
Test Vector
Generator
(Test Bench)
Checker
Module
(Test Bench)

Figure 3

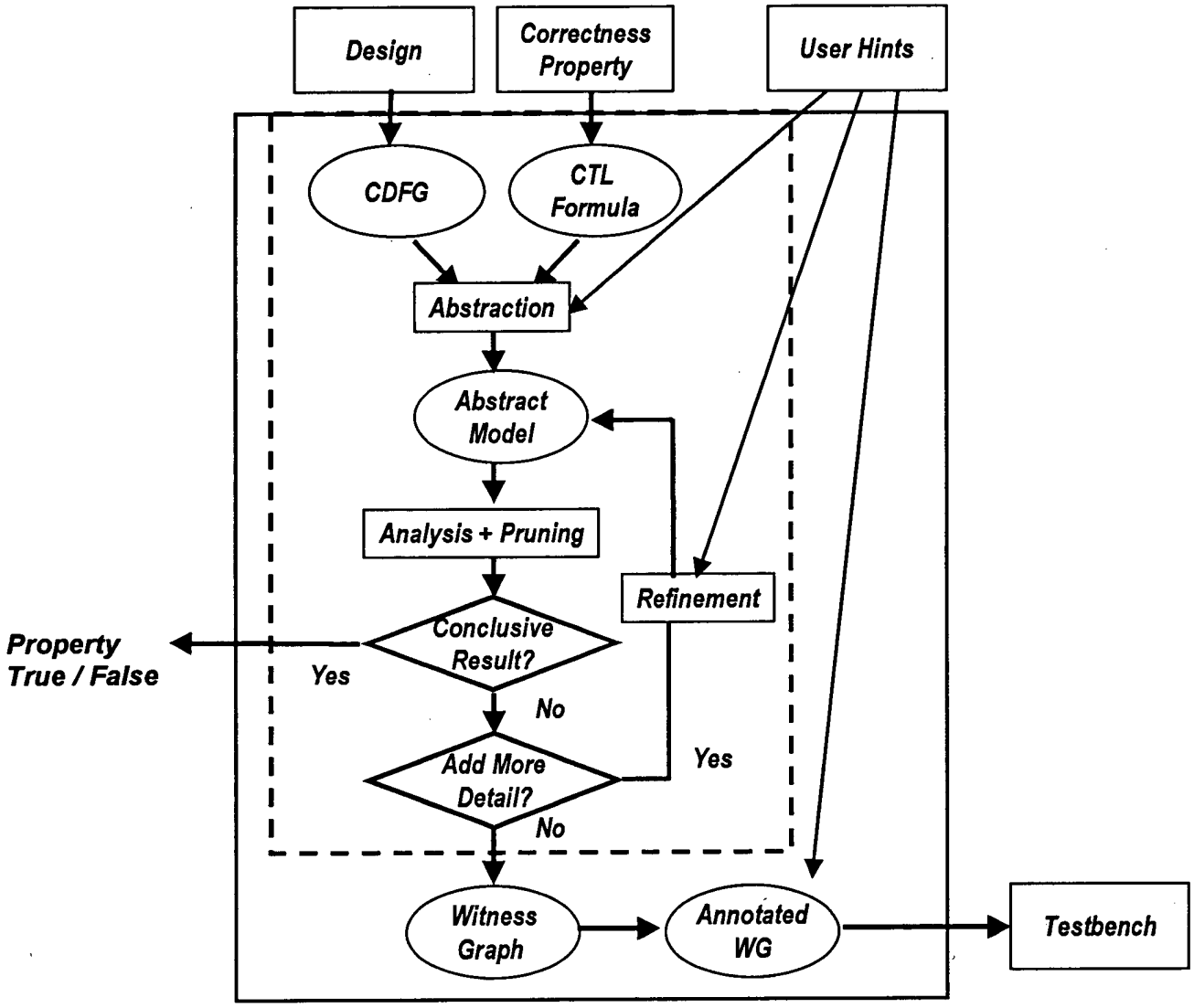
Design
Correctness Property
User Hints
CDFG
CTL Formula
Abstraction
Abstract Model
Analysis + Pruning
Refinement
Conclusive Result?
Property True / False
Yes
No
Add More Detail?
Yes
No
Witness Graph
Annotated WG
Testbench

Figure 4

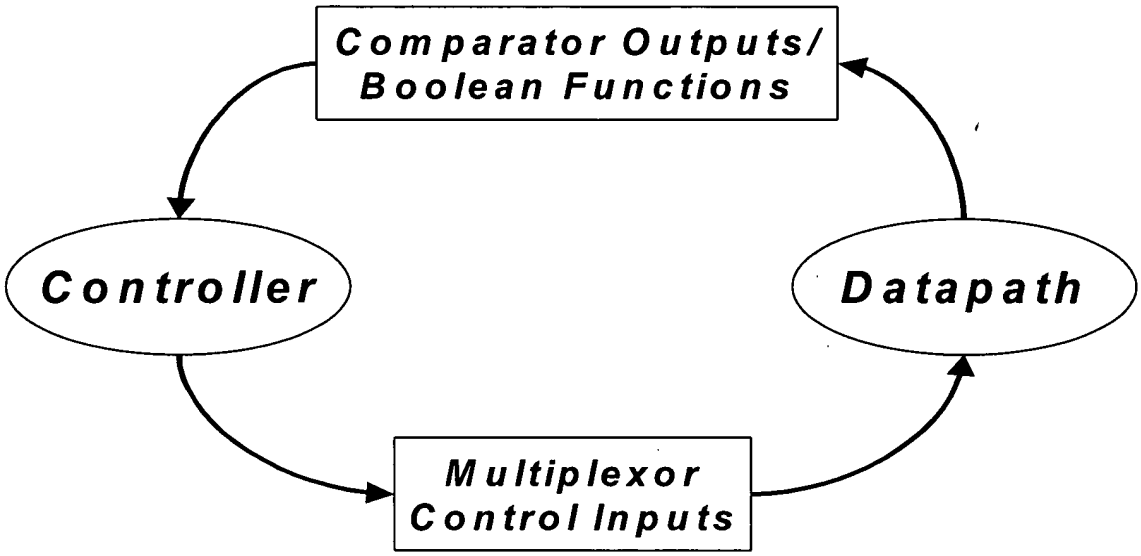
Comparator Outputs/
Boolean Functions
Controller
Datapath
Multiplexor
Control Inputs

Figure 5

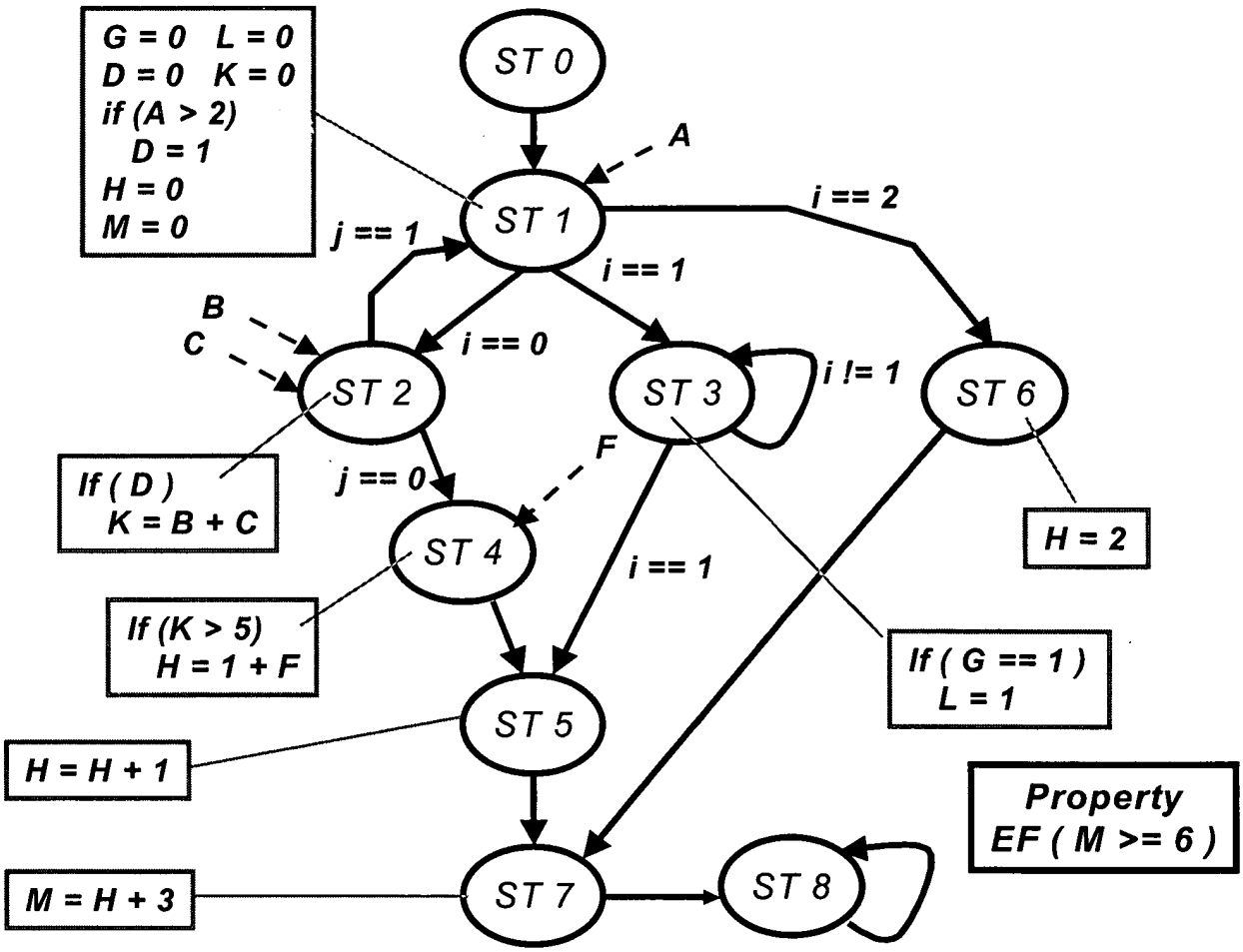
G = 0 L = 0
D = 0 K = 0
if (A > 2)
D = 1
H = 0
M = 0
ST 0
A
ST 1
j == 1
i == 2
i == 1
B
C
i == 0
ST 2
ST 3
i != 1
ST 6
If (D)
K = B + C
j == 0
F
ST 4
H = 2
If (K > 5)
H = 1 + F
i == 1
If (G == 1)
L = 1
ST 5
H = H + 1
Property
EF (M >= 6)
M = H + 3
ST 7
ST 8

Figure 6

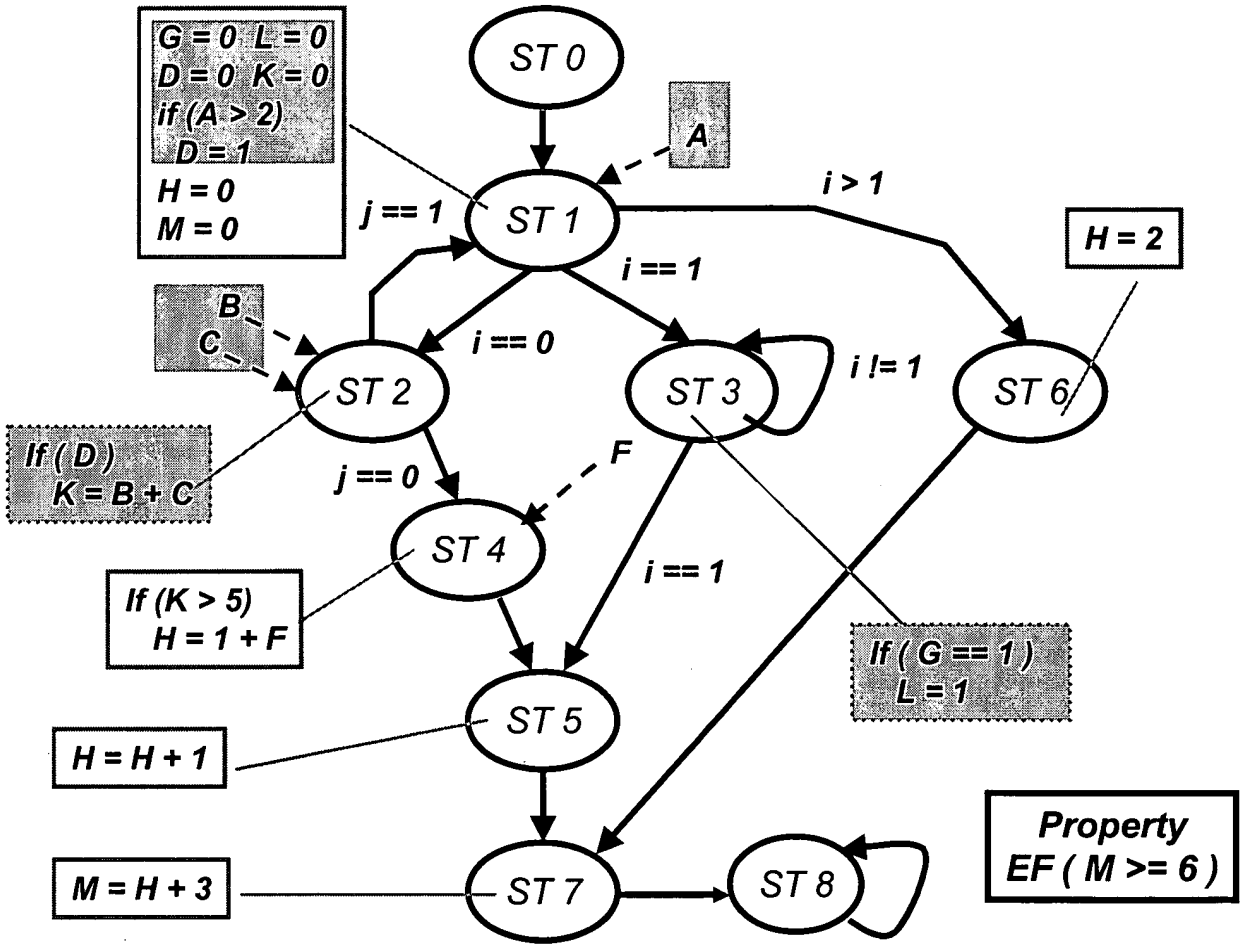
G = 0 L = 0
D = 0 K = 0
if (A > 2)
D = 1
H = 0
M = 0
ST 0
A
ST 1
j == 1
i > 1
H = 2
i == 1
B
C
i == 0
ST 2
ST 3
i != 1
ST 6
If (D)
K = B + C
j == 0
F
ST 4
If (K > 5)
H = 1 + F
i == 1
If (G == 1)
L = 1
ST 5
H = H + 1
Property
EF (M >= 6)
M = H + 3
ST 7
ST 8

Figure 7

```
mc_for_sim(model m, ctlFormula f) {
 ctlFormula f1, f2, negf;
 states upper,upper1,upper2=NULL,negative=NULL;
 // handle subformulas recursively

 if (f1 = leftChild(f)) {
   mc_for_sim(m,f1);
   upper1 = get_upper(f1);
 }

 if (f2 = rightChild(f)) {
   mc_for_sim(m,f2));
   upper2 = get_upper(f2);
 }

 // case analysis on operator at this level
 switch(type(f)) {
      case TRUE: upper = ALL; break;
      case FALSE: upper = NULL; break;
      case ATOMIC: upper = mc_atomic(m,f); break;
      case NOT: upper = complement(upper1); break;
      case AND: upper = and(upper1,upper2); break;
      case OR: upper = or(upper1,upper2); break;
      case EX: case EF: case EU: case EG:
        upper = mc_etype(upper1,upper2); break;
      default: // A-type operators left
        switch(type(f)) {
        case AX: upper = mc_ex(upper1); break;
        case AF: upper = mc_ef(upper1); break;
        case AU: upper = mc_eu(upper1,upper2); break;
        case AG: upper = mc_eg(upper1); break;
      }
   // compute negative sets also
   negf = negate(f);
   mc_for_sim(m,negf);
   negative = and(upper,get_upper(negf)); break;
 }

 // associate the sets with f
 associate(f, upper, negative);
}
```

Figure 8

```
check_mc (model m, ctlFormula f)
{
  mc_for_sim(m, f);
  if (initState(m) ∉ get_upper(f))
    result = PROPERTY_FALSE;
  else if (A-type(f) &&
   initState(m) ∉ get_negative(f))
    result = PROPERTY_TRUE;
  else
    result = INCONCLUSIVE;
  return result;
}
```

Figure 9

```
mark_witness_top(model m, ctlFormula f)
{
  reachable = compute_reachable(m,initState(m));

  switch(type(f)){
       case AX: case AF: case AU: case AG:
         witness_top= and(get_negative(f),reachable);
         break;
       default:
         witness_top= and(get_upper(f),reachable);
  }

  mark_states(witness_top);
  mark_witness_rec(m, f, witness_top);
}
```

Figure 10a

```
mark_witness_rec(model m, ctlFormula f, states careSet)
{
 states witness, negWitness, subWitness;
 //   associate witness set for f
 witness = and(get_upper(f),careSet);
 associate_witness(f,witness);
 //   recursive calls with modified careSets
 switch(type(f)) {

      case TRUE:case FALSE:case ATOMIC:case NOT:
        break;

      case AND: case OR:
      case EF: case EU: case EG:
        mark_witness_rec(m,leftChild(f),witness);
        if (rightChild(f) != NULL)
          mark_witness_rec(m,rightChild(f),witness);
        break;

      case EX:
        subWitness = compute_image(m, witness);
        // mark additional states
        mark_states(subWitness);
        mark_witness_rec(m,leftChild(f),subWitness);
        break;

      case AX: case AF: case AU: case AG:
        negWitness = and(get_negative(f),careSet);
        associate_neg_witness(f,negWitness);
        mark_witness_rec(m,negate(f),negWitness);
        break;
 }
}
```

Figure 10b

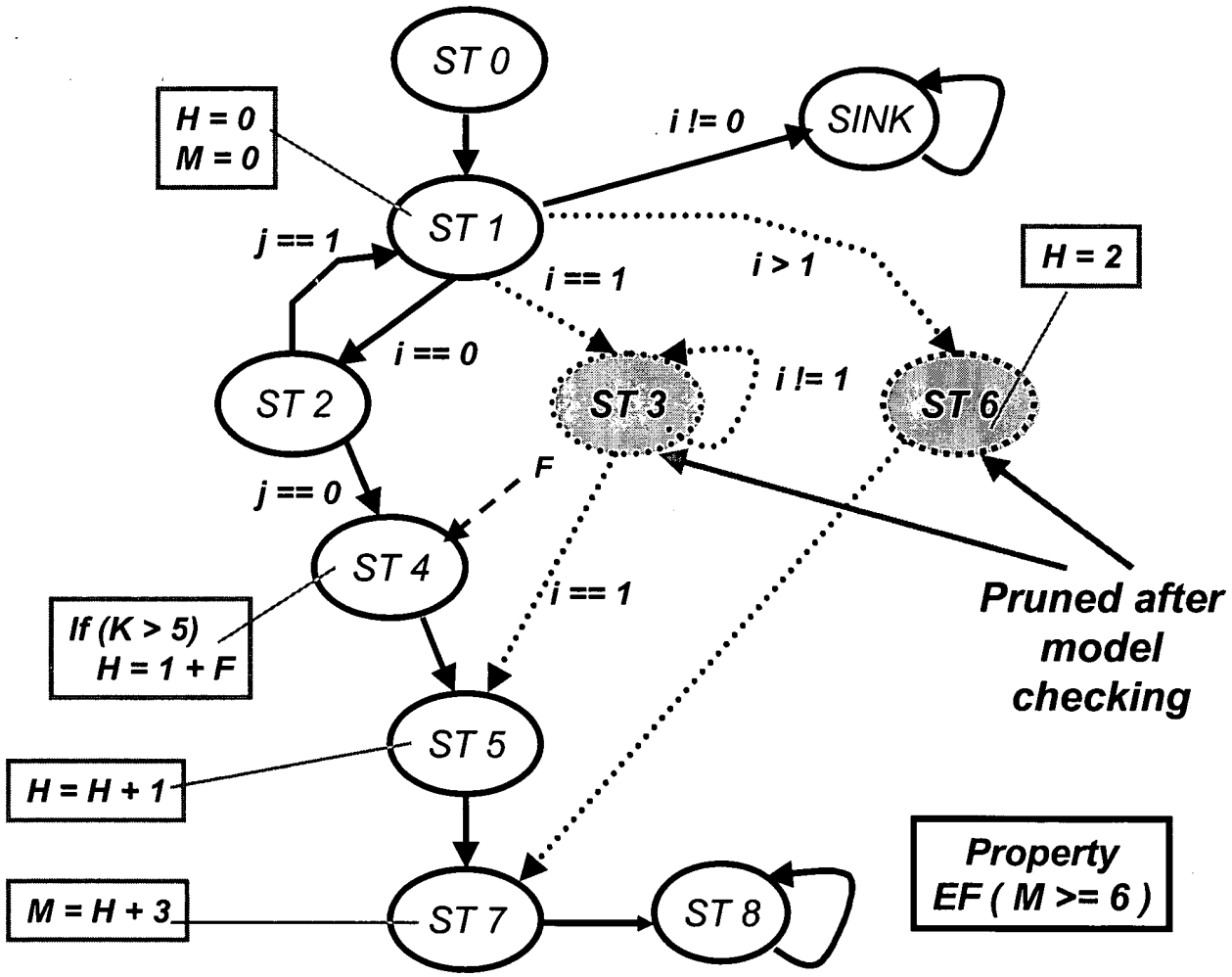
ST 0
H = 0
M = 0
i != 0
SINK
ST 1
j == 1
i == 1
i > 1
H = 2
i == 0
ST 2
ST 3
i != 1
ST 6
F
j == 0
ST 4
i == 1
If (K > 5)
H = 1 + F
Pruned after
model
checking
ST 5
H = H + 1
Property
EF (M >= 6)
M = H + 3
ST 7
ST 8

Figure 11

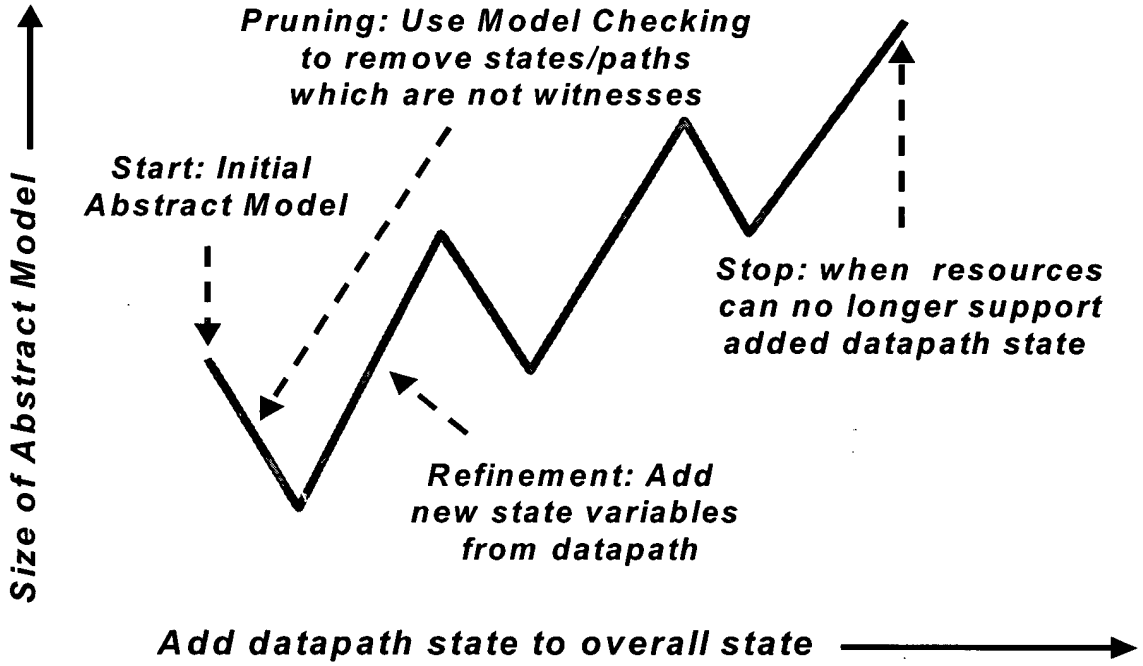
Pruning: Use Model Checking
to remove states/paths
which are not witnesses
Start: Initial
Abstract Model
Stop: when resources
can no longer support
added datapath state
Refinement: Add
new state variables
from datapath
Size of Abstract Model
Add datapath state to overall state

Figure 12

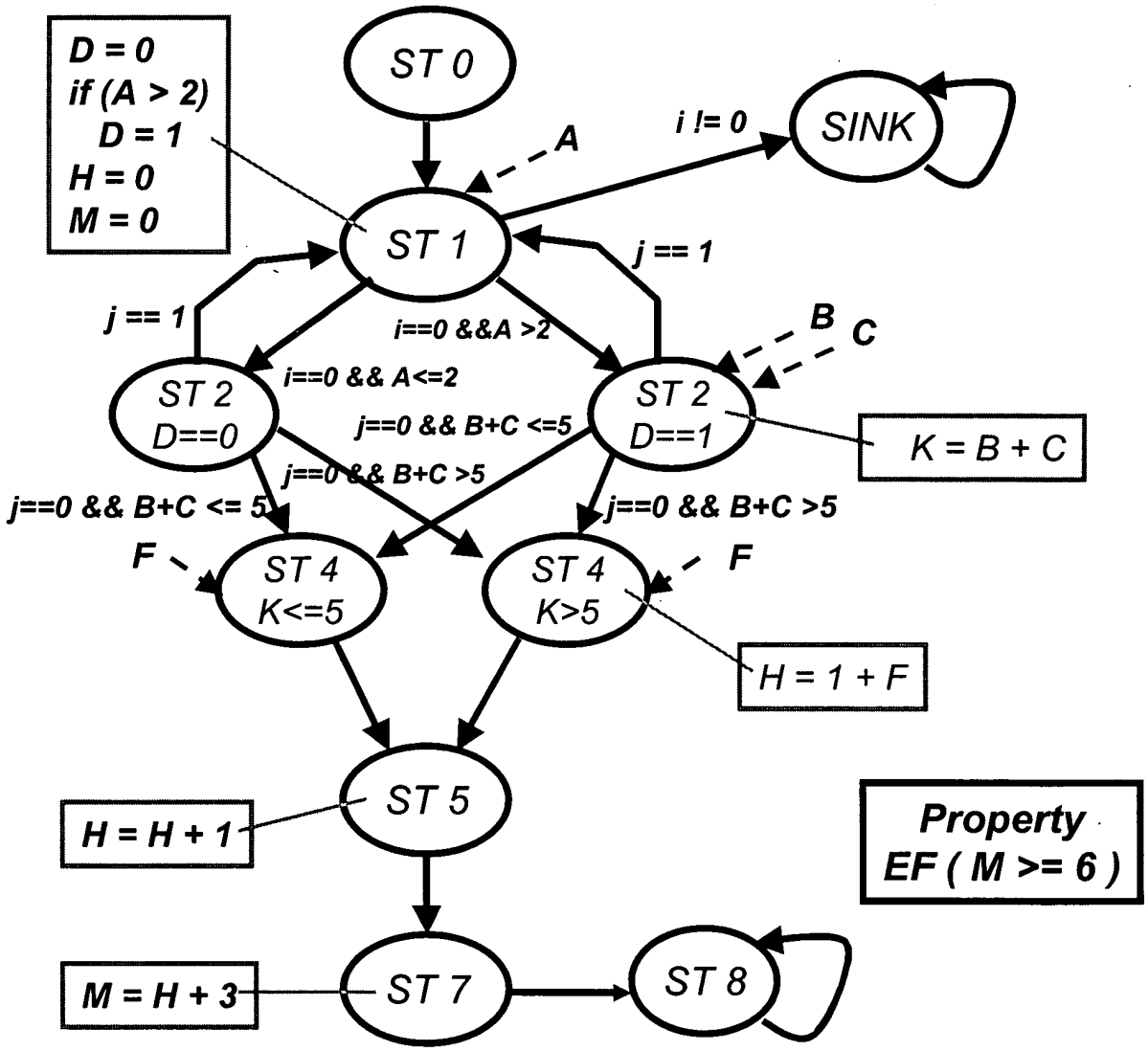
D = 0
if (A > 2)
D = 1
H = 0
M = 0
ST 0
A
i != 0
SINK
ST 1
j == 1
j == 1
i==0 &&A >2
B
C
ST 2
D==0
i==0 && A<=2
ST 2
D==1
K = B + C
j==0 && B+C <=5
j==0 && B+C >5
j==0 && B+C <= 5
j==0 && B+C >5
F
ST 4
K<=5
ST 4
K>5
F
H = 1 + F
ST 5
H = H + 1
Property
EF (M >= 6)
M = H + 3
ST 7
ST 8

Figure 13

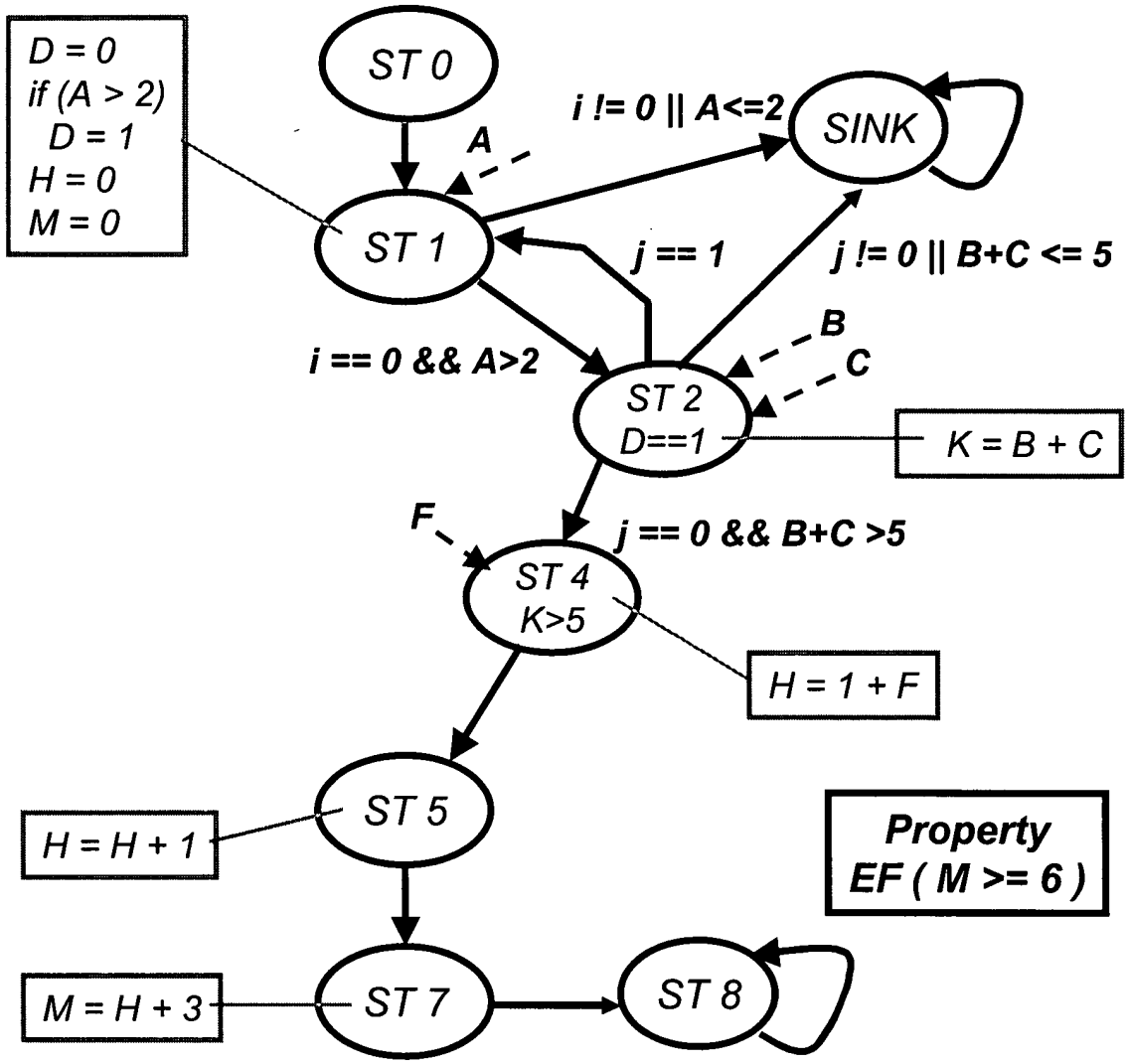
D = 0
if (A > 2)
D = 1
H = 0
M = 0
ST 0
A
i != 0 || A<=2
SINK
ST 1
j == 1
j != 0 || B+C <= 5
B
C
i == 0 && A>2
ST 2
D==1
K = B + C
F
j == 0 && B+C >5
ST 4
K>5
H = 1 + F
ST 5
H = H + 1
Property
EF (M >= 6)
M = H + 3
ST 7
ST 8

Figure 14

```
witness_sim(design d, ctlFormula f, state s)
{
     states w, w1;
     int result, neg_result;
     w = get_witness(f);
     w1 = get_witness(leftChild(f));
     //   case analysis on operator at this level

     switch(type(f)) {

          case TRUE: result = SUCCESS; break;

          case FALSE: result = FAILURE; break;

          case ATOMIC: result = satisfies(s,f); break;

          case NOT: result=satisfies(s,negate(f));break;

          case AND:
               result = witness_sim(d,leftChild(f),s);
               if (result==SUCCESS)
                    result = witness_sim(d,rightChild(f),s);
               break;

          case OR:
               result = witness_sim(d,leftChild(f),s);
               if (result==FAILURE)
                    result = witness_sim(d,rightChild(f),s);
               break;

          case EX:
               foreach state t, abs(t)∈w1, {
                    if (exists_transition(s,t)){
                         result = witness_sim(d,leftChild(f),t);
                         if (result==SUCCESS) break;
                    }
               }
               break;
```

Figure 15a

```
case EF:
     foreach state t, abs(t)∈w1, {
          if (path = find_a_path(s,t)){
               result = witness_sim(d,leftChild(f),t);
               if (result==SUCCESS) break;
          }
     }
     break;

case EU:
     result = witness_sim(d,rightChild(f),s);
     if (result==FAILURE){
          mark(s,f);
          result = witness_sim(d,leftChild(f),s);
          if (result==SUCCESS)
               foreach unmarked state t, abs(t)∈w {
                    if (exists_transition(s,t)){
                         result = witness_sim(d,f,t);
                         if (result==SUCCESS) break;
                    }
               }
     }
     break;

case EG:
     result = witness_sim(d,leftChild(f),s);
     if (result==SUCCESS){
          mark(s,f);
          if (exists_transition_to_marked(s,f))
               result = SUCCESS;
          else
               foreach unmarked state t,abs(t)∈w {
                    if (exists_transition(s,t)){
                         result = witness_sim(d,f,t);
                         if (result==SUCCESS) break;
                    }
               }
     }
     break;
```

Figure 15b

```
        case AX: case AF: case AU: case AG:
             if (abs(s) ∉ get_neg_witness(f))
                  result = SUCCESS;
             else {
                  // generate counter-example for !f
                  neg_result = witness_sim(d,negate(f),s);
                  result = (neg_result == SUCCESS) ?
                       FAILURE : SUCCESS;
             }

    } // end switch
    return result;
}
```

Figure 15c

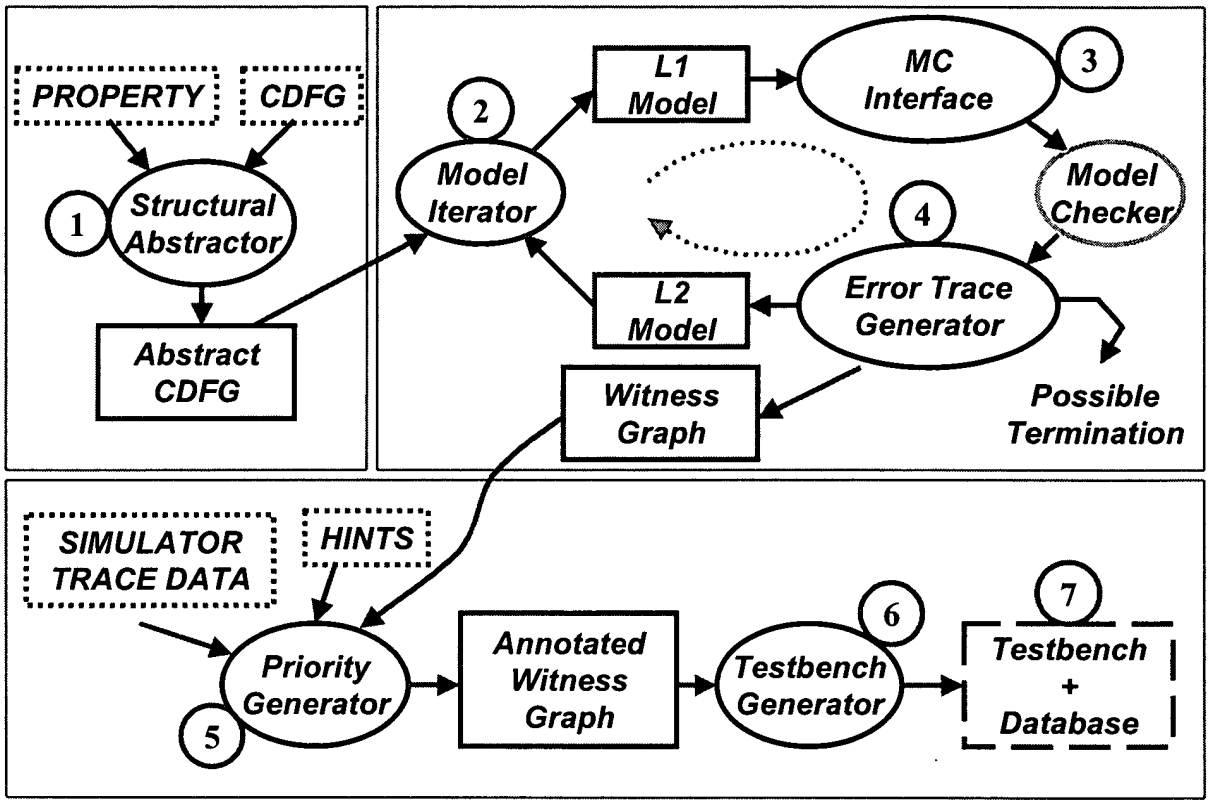
PROPERTY
CDFG
1
Structural Abstractor
Abstract CDFG
2
Model Iterator
L1 Model
MC Interface
3
Model Checker
4
Error Trace Generator
L2 Model
Witness Graph
Possible Termination
SIMULATOR TRACE DATA
HINTS
Priority Generator
5
Annotated Witness Graph
6
Testbench Generator
7
Testbench + Database

Figure 16

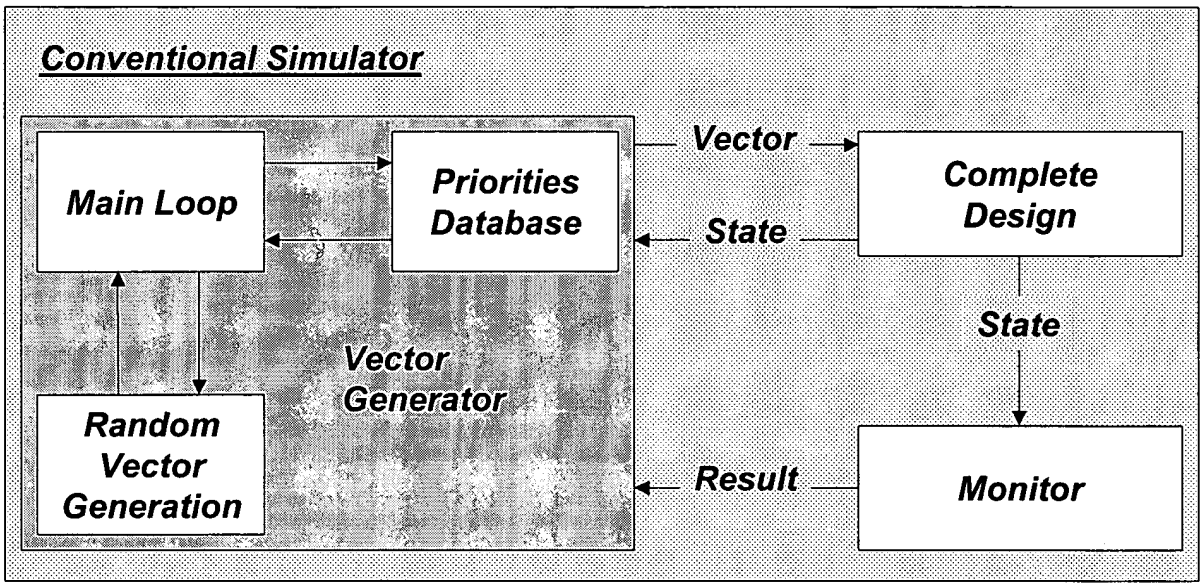
Conventional Simulator
Main Loop
Priorities Database
Vector Generator
Random Vector Generation
Vector
State
Result
Complete Design
State
Monitor

Figure 17

```
Testbench(){
...
 do {
  determine current state of design;
  determine abstract state from current state;
  query database for desirable transition;
  if (input vector NOT in database) {
L1: input vector = random vector;
    if (input vector satisfies condition) {
      simulate input vector;
      if (next abstract state != desired) {
        roll back simulation one cycle;
        go to L1;
      }
    }
  }
 } while (property is not yet proved/disproved);
...
}
```

Figure 18